|  |  |
| --- | --- |
| Lab | Friday 9AM |
| Student No | 16725829 |
| Registered Programme | DN201 Data Sci |
| Date | 02/11/2018 |

**Exercise 1**

Compute the following small ridge regression problem **by hand on paper**.

Let x1=(2,3), x2=(5,-1), x3 = (0,-2) be three features in the positive class and

x4=(3,-2), x5 = (4,3), x6=(5,2) be three feature vectors in the negative class.

Take lambda=1.0;

1. Write down the feature matrix X and the output vector y:
2. Compute the 2x2 matrix X’\*X
3. Compute the vector b = X’\*y;
4. Compute the 2x2 matrix A = (X’\*X + lambda\*I)
5. Compute the B = inv(A);
6. Compute w = B\*b
7. If the test set contains 2 positive class feature vectors (0,1) and (-1,-1) and two negative class feature vectors (2,3) and (4,2), compute the value of the hypothesis at each of these test feature vectors:
8. Hence, compute the **accuracy**, **precision**, **recall** and **specificity** of the classifier on this test set.

Note that when then where

**Input your answers into the Moodle Quiz associated with this Lab.**

**Exercise 2**

From the Moodle web-site, download the file features.txt into a working directory, say Desktop/Lab6. In Matlab, change into this directory and read the data in the file into a matrix X:

>> cd Desktop/Lab6

>> X = load(‘features.txt’);

X contains a set of two-dimensional feature vectors, along with their class labels (either +1 or -1). Let’s firstly remove the class labels from X and put them into a label vector called Y:

>> y = X(:,end);

>> X = X(:, 1:(end-1));

Now let’s create a training and a test set. Choose 80% of the data for training and the remaining 20% for testing:

>> N = size(X,1);

>> perm = randperm(N); % this gives a permutation of the

% indices 1,…, N

>> Ntrain = ceil(N\*0.8); % choose 80% of N

>> Ntest = N – Ntrain; % number of test samples

>> Xtrain = X(perm(1:Ntrain), :); % choose Ntrain randomly-

% selected samples

>> Xtest = X(perm((Ntrain+1):end),:); % the remaining samples

% go in the test set

>> ytrain = y(perm(1:Ntrain));

>> ytest = y(perm((Ntrain+1):end));

Let’s plot the training set, showing the two different classes in different colours.

>> neg\_class = (ytrain==-1);

>> pos\_class = (ytrain==1);

>> figure; hold(‘on’);

>> plot(Xtrain(neg\_class,1),Xtrain(neg\_class,2), ‘b.’);

>> plot(Xtrain(pos\_class,1),Xtrain(pos\_class,2),’r.’);

Usually, we add a dummy feature of all ones to represent the bias (see the lecture notes). Let’s do that:

>> Xtrain = [ones(Ntrain,1), Xtrain];

>> Xtest = [ones(Ntest, 1), Xtest];

If you type Xtrain now, you should see that it is a set of 3-dimensional feature vectors, with the first feature set to 1:

>> Xtrain

Let’s try to do Machine Learning by hand: Use the gline function to place a line on the plot that best (to your eye) divides the red dots from the blue dots

>> gline

Let’s pick two points on that line, in order to work out the equation of the chosen line. We’ll put those points into a 2x2 matrix P:

>> P = zeros(2,2);

Now pick two points on the line using getpts. After the call to getpts, use your mouse to pick two points on your chosen line.

>> [P(:,1),P(:,2)] = getpts

Compute the equation of the line[[1]](#footnote-1) in the form w\_0 + w\_1 x + w\_2 y = 0, where

>> w\_2 = 1;

>> w\_1 = - (P(2,2)-P(1,2))/(P(2,1)-P(1,1));

>> w\_0 = -w\_1 \* P(1,1) - P(1,2);

This line is the decision boundary, but we need to be careful to get the sign right. Choose a point on your figure that’s definitely on the side of the line where you see most blue dots i.e. the most points that belong to the ‘negative’ class, e.g. point (0,3). Since the blue dots correspond to the negative class, we need points in this region to evaluate to a negative number, when inserted in the decision line equation. Check it out

>> w\_0 + 0\*w\_1 + 3\*w\_2

If the **answer is** **positive**, then you need to **change the sign** of the weights by doing:

>> w\_0=-w\_0; w\_1 = -w\_1; w\_2 = -w\_2;

Gather all of these weights into a weight vector:

>> w = [w\_0;w\_1;w\_2]

Now, whenever w\_0 + w\_1\*Xtrain(i,1) + w\_2\*Xtrain(i,2)>0 we predict the ‘positive’ class, otherwise we predict the ‘negative’ class. We can do this succinctly in Matlab, as follows:

>> yhat = (Xtrain\*w > 0);

>> yhat = 2\*yhat – 1; % change 0/1 to -1/+1

Work out some metrics:

1. **Accuracy** is the number of predictions we got correct as a percentage of the total number of instances in the database:

>> accuracy = sum(yhat == ytrain)/length(yhat);

1. **Precision** is the number of the positive class predictions that truly belong in the positive class as a percentage of the total number of positive class predictions.

>> precision = sum(yhat == 1 & y==1)/sum(yhat==1)

1. Can you write an expression to compute the **recall**, the number of positive class predictions that truly belong in the positive class as a percentage of the total number of positive class instances?
2. Write an expression to compute the **specificity -**  the percentage of the actual negative class instances that were correctly identified.

Compute the accuracy, precision, recall and specificity on the **test data**, Xtest and ytest.

Note that usually performance is better on the training set, though this may not be the case for the small dataset that we’ve selected.

The true test of performance is how well the classifier performs on unseen, test data.

Finally, load the file **testfile.txt**, which contains more samples on which to test your classifier. Using the weight vector w that you created above, evaluate the accuracy, precision, recall and specificity of your classifier, **computed on the testfile.txt** data.

**Input your answers into the Moodle Quiz associated with this Lab.**

**Exercise 3**

Next, use Ridge Regression to automatically find a decision boundary in the above problem.

Compute the matrix X’\*X. This is a 3x3 matrix, as there are 3 features (including the bias) in each feature vector.

>> A = Xtrain’\*Xtrain;

Add to A some multiple of the identity matrix. The eye(3) is the Matlab function to produce a 3x3 identity matrix:

>> lambda = 1.0;

>> A = A + lambda \* eye(3)

Multiply the ytrain vector by X’

>> b = Xtrain’\*ytrain;

Finally compute the weights by multiplying b by the inverse of A:

>> w\_rr = inv(A)\*b;

**Input your answer into the Moodle Quiz associated with this Lab.**

Compare the ridge regression solution with the one you constructed by hand by plotting the ridge regression decision boundary:

Pick some co-ordinates along the boundary:

>> coords\_x = -1:0.01:1;

>> coords\_y = (-w\_rr(2)\*coords\_x – w\_rr(1))/w\_rr(3);

>> plot(coords\_x,coords\_y,’k’);

1. Recall that the equation of a line, with two points and is given by where ) and . The two required points are =(P(1,1), P(1,2)) and =(P(2,1), P(2,2)), which are obtained from the call to getpts [↑](#footnote-ref-1)